第3章

ビデオ・エフェクト処理を実現する演算回路の最適化設計

――回路規模を小さくするための着眼点と道筋の見極めかた

蓮鳥明日香

ここでは，テレビ放送などでおなじみのディジタル・ビデオ・エフェクト処理を例に挙げて，回路仕様の決めかたや処理手順などを解説する．この回路では，リアルタイムで画像の変形処理を行うため，複数の関数を取り扱う．FPGAのリソースを抑え，ある程度の精度を維持するには，システム・レベルの検討が重要，と筆者は言う．（編集部）

ディジタル・ビデオ・エフェクト（DVE）は，ビデオ画像に変形効果を加える手段として，現在，主流となっています．例えば，図1に示すようなビデオ・エフェクトによる処理の中には，テレビなどでおなじみのものもあると思います．このような効果は，x-y平面にある画像を原画像としてx，y方向に変形をかけることで得られますが，画像をディジタル・データとすることで，メモリ上に書き込んだ原画像を自在に読み出すことができるようになり，これによってさまざまな変形を生み出すわけです．

パソコン向けのソフトウェアの中に上記と似たような機能を備えているものがあることからもわかるように，こうした効果はあえてハードウェア化しなくてもソフトウェアで実現できます．しかし，ニュースやスポーツなどのテレビの生放送では，ハードウェアによるビデオ・エフェクト処理が欠かせないものとなっています．その理由は，リアルタイムで十分な画質を保って効果を得るには，今のところハードウェアに頼らざるをえないからです．ハードウェアによるビデオ・エフェクト処理は，多くの場合，画面の切り換えに使われることが多いようです．例えば，野球中

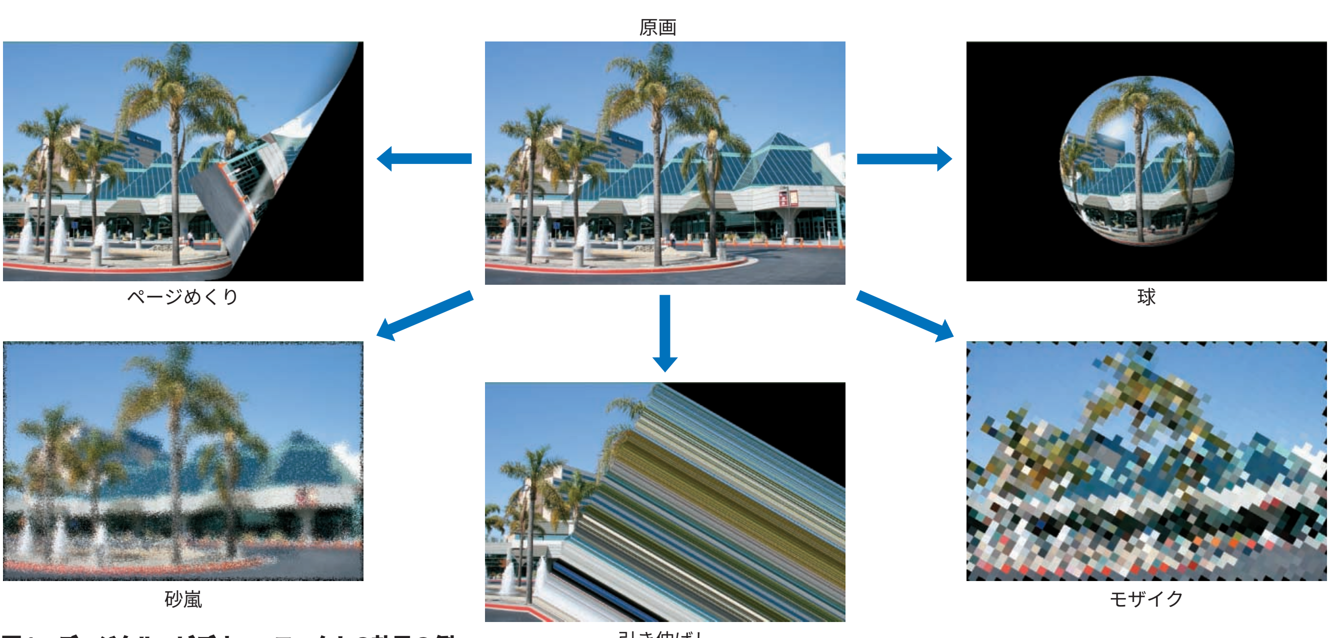


図1　ディジタル・ビデオ・エフェクトの効果の例
中にはテレビでおなじみの処理もある．

継などで，ホームランがスタンドに入るシーンから次のバッターの生映像へ，ページをめくるように切り換えるといったぐあいです．**図1**のページめくり効果において，黒い背景となっている部分に別の画像がはまっているイメージと考えればよいでしょう．

ここでは，ディジタル・ビデオ・エフェクトのメモリ読み出しアドレス生成部を例にとって，数値演算回路の最適化事例を紹介します．なお，ここで言う最適化とは，簡潔で規模の小さな実現構造を指します．

● 保守性や配置配線性の良い「スマートな設計」を目指す

ディジタル・ビデオ・エフェクトのアドレス生成部は，数値演算の塊といっていいでしょう．しかし，機能別に見ていくと，目的ごとに性格の異なる部分があります．本稿では，ここに着目して実現手段を選択することで回路の最適化を図り，回路規模を抑制できることを示します．

ただし，回路規模を気にするあまり，回路の共用を極度に進めると，非常に難解で保守性の悪い回路になってしまいます．また，そのような回路は往々にしてターゲットとなるFPGA上で配置配線性の悪い回路になりがちです．力技も時には必要ですが，まずはスマートな実現方法を目指しましょう．一歩引いて見ることで物事を的確にとらえることができるのは，回路設計でも同じです．

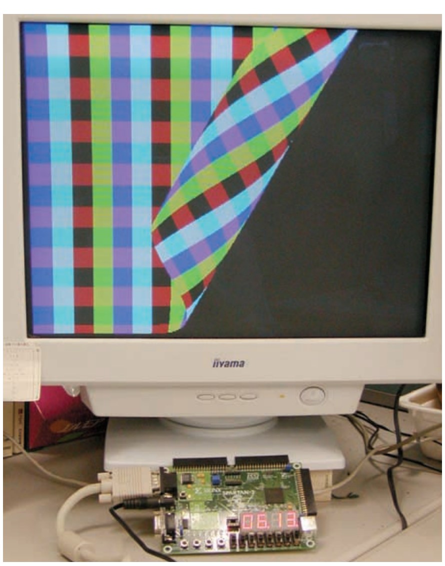


写真1　FPGAに実装して動作させたようす

目標は，アドレス生成部とビデオ信号処理部を40万システム・ゲートのFPGA（XC3S400）に実装するとしたが，写真の実機動作の場合はアドレス生成部のみを実装したので20万システム・ゲートのFPGA（XC3S200）で60％の使用率だった．使用したFPGAボードは，Xilinx社の「Spartan-3 Starter Kit」に付属しているもの．

また，効率の良い実装に必要とされるものとして，ハードウェアとソフトウェアの共存があります．例えば，ビデオ・アプリケーションでは，1フレーム（あるいは1フィールド）に1回更新されればよい制御パラメータが数多くありますが，これらを算出するのにハードウェアを適用するのは好ましくありません．ソフトウェアに演算の余裕がある限りソフトウェアで処理するのが最適なシステムの構造であると筆者は考えます．余分なハードウェア・リソースはとことん排除しましょう．

ここでは要求仕様が以下のものとして設計を始めます．

- ビデオ・フォーマット：NTSC
- ターゲット・デバイス：米国Xilinx社のFPGA「Spartan-3（XC3S400-4）」（システム・ゲート数40万）

写真1に，実際のFPGAに実装して画面に表示したようすを示します．

本稿の説明の流れは以下のとおりです．

1) **ディジタル・ビデオ・エフェクトの原理**（何を作るのか，基本的な部分を押さえる）
2) **各効果別の実現**（各効果を実現するために必要な演算を数学的に理解する）
3) **各効果の結合**（別々の機能を持つ効果を統合させる際の考えかたを理解する）
4) **トップ・レベルの構造構築**（スマートな構造の構築や規模の抑制にあたっての基本的な指針を理解する）
5) **サブモジュール単位の具体的な回路構造**

ここで重要な箇所は**2)**と**3)**です．一見無関係な複数の機能が一つにまとまっていく過程を読み取っていただければと思います．

1 効果の原理――逆写像が必要

図2にディジタル・ビデオ・エフェクトの効果の基本原理を示します．なお，ここでは，元となる原画像を「ソース画面」，変形後の画像を「デスティネーション画面」と呼ぶことにします．ディジタル・ビデオ・エフェクトを直感的に考えると，「ソース画面の画素データをデスティネーション画面へ貼り付けるために必要な変換（写像）を求める」ということになりそうです．

しかし，実際に私たちが見るのは変形後の画面になるので，デスティネーション画面の上をラスタ・スキャンして画像を見ることになります．このラスタ・スキャンしてい

る座標に貼り付けるソース画面上の画素データの座標を逆変換（逆写像）によって求める，という方法が一般的に用いられます．少々まわりくどくなりましたが，ここでは「逆写像が必要である」ということだけ覚えておいてください（詳しく知りたい方はコンピュータ・グラフィックスなどの参考書を参照のこと．参考文献(1)はお勧め）．

図3に，これを実現するためのブロック図を示します．今回着目するのは図の中で黄色く示したブロックです．なぜアドレス生成回路と呼ばれるかというと，メモリの読み出しアドレス（逆写像アドレス）を生成するからです．

なお，今回ここで紹介するのは2次元(2D)のディジタル・ビデオ・エフェクトです．陰影を付けることで立体的に見せていますが，あくまでも正面から見た平面を表現しているに過ぎません．

2 最初の一歩は「数学的アプローチ」

ここからは，各効果を実現するにあたって必要な演算を検討します．あくまでも算術的なアプローチとなりますが，この段階で複数の機能をいかに統合するかを検討しておくことは，非常に重要です．これをやっておかないと，最悪の場合，機能ごとに一つ一つ回路を作って，出力をマルチプレクスする，といった非常にみっともない結果になりかねません．

● ページめくり効果――四つの領域に分けて考える

図4に，ページめくり効果の実現方法を示します．制御パラメータは次の三つです．

- めくり半径：R
- めくり量：$\varDelta x$
- めくり方向：θ_r

図4(c)は画面を上から見た図で，$\varDelta x$とθ_rを定義しています．図4(d)はめくり方向の真横から見た図で，Rと座標の関係を示しています．視点から見ためくられた形がデスティネーション画面上に描画されており，このめくりを戻してやったときに対応する点がソース画面上の画素データになります．例えば，図4(d)の点P0はめくりの起点となる場所で，点P1は視点から見て点P0の右にRだけ離れた位置に見えます．めくりを戻していくと，ソース画面上の対応する点P1 'はP0から円周の1/4の距離にあることになります．

このようにすると，領域を四つに場合分けして考えればよいことになります．めくる前の領域をarea0とし，あとは90 °ずつarea1，area2と続き，area3では図4(a)のページめくり1と図4(b)のページめくり2で別処理になります．ページめくり2の場合，area4まで考える必要がありそうですが，この部分は不可視ですので必要ありません．

これをまとめると，表1のようになります．なお，表の中でx，yと表記されているのは，x-y平面上の座標そのものではなく，図4(c)におけるめくり中心線をy軸，効果のかかる方向をx軸とした直交座標によるものです．効果のかかる方向がx軸に一致し，めくりの中心が原点を通ると

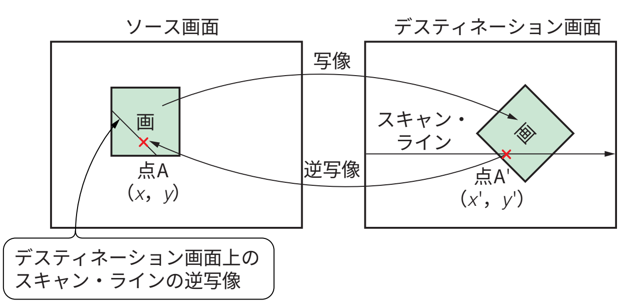


図2　ディジタル・ビデオ・エフェクト実現の基本原理
ラスタ・スキャンしている座標に貼り付けるソース画面上の画素データの座標を逆変換（逆写像）によって求める，という方法が一般的に用いられる．

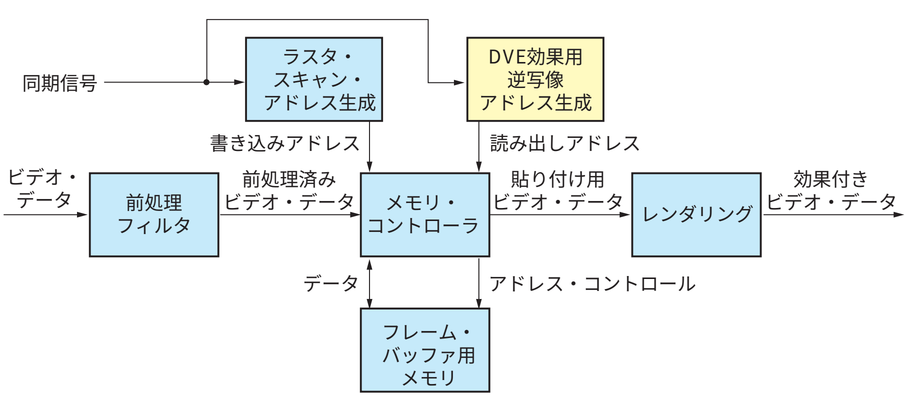


図3　ディジタル・ビデオ・エフェクト実現の基本ブロック図
必要に応じて前処理フィルタをかけたソース画面を，一度フレーム・バッファ・メモリに書き込み，アドレス生成回路によって生成された逆写像アドレスによって画素データを読み出し，走査線に貼り付けること（レンダリング）でデスティネーション画面を構成する．

きに限り，本来のx-y座標と一致します．

ここまでできれば，x軸に沿ってめくることができます．しかし，紙をめくる効果なので，任意の方向にめくれるようにしておかなければなりません．そこで，図4(c)の原点周りに回転する操作が必要になります．

以上のことから，次の二つの工程が必要になります．

- z軸方向の情報をx-y平面上に展開していく
- めくり方向に回転させる

しかし，単純に二つの工程を直列に並べたのでは全体が回転してしまい，ここで目指すものとは異なってしまいます．例えば，めくられていない部分に回転操作があってはならないのです．したがって，z軸方向の情報をx-y平面上にめくり方向に沿って展開していく必要があります．具体的に言えば，以下のような手順になります．

- x-y座標よりめくり中心線からの距離を算出（効果のかかる方向への距離に変換）
- めくり中心線からの距離をxとして，表1による演算を適用（効果のかかる方向をx軸としてめくり変換）
- 表1による演算結果をx-y成分に分離し，めくり中心線を基準に加算（本来のx-y平面上の値に換算）

ここまででわかることは，「なんだか手間がかかりそうだ」ということと，「イメージがとらえにくい」ということでしょうか．少なくとも，筆者が初めてこのアプリケーションを扱ったときは，かなり混乱したのを覚えています．ほかの効果とも共存しなければならないのに，ページめくりだけで十分ややこしいことになってしまったからです．逆にこれをスマートにまとめるところに，設計のおもしろさがあるとも言えます．

ここで補足しておきますが，ページめくり効果を上から見た場合，可視の部分と不可視の部分が存在します．コンピュータ・グラフィックスの分野では「陰面消去」などと呼ばれ，処理効率を上げるために重要な問題として取り上げ

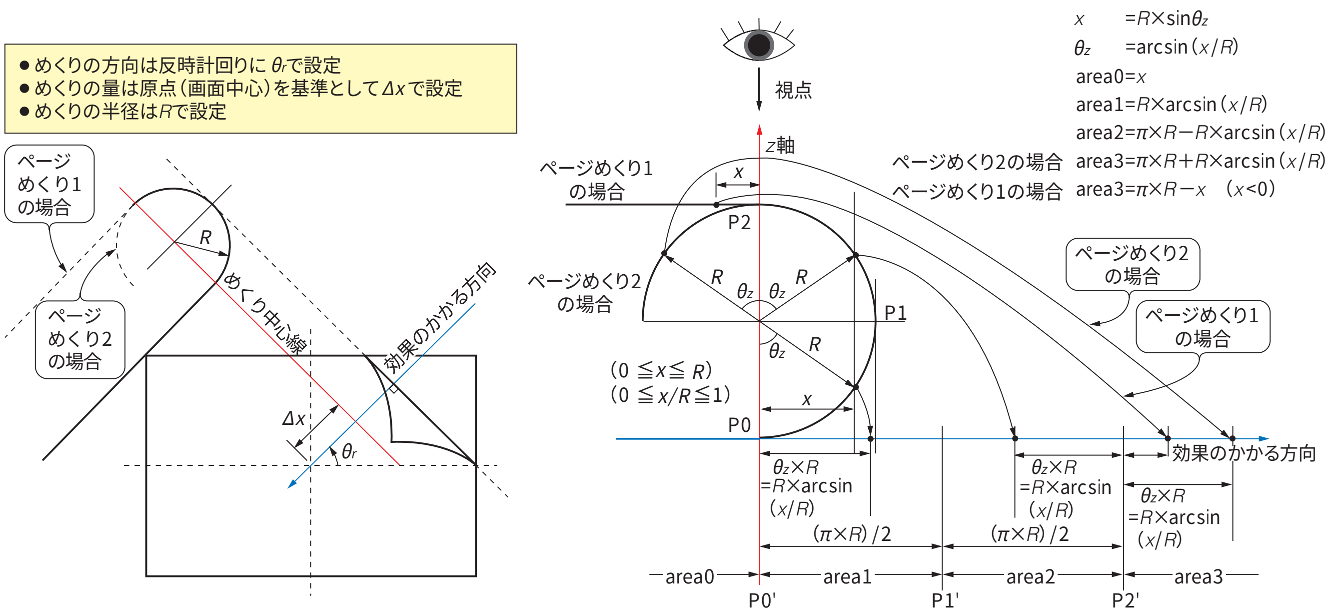
表1　ページめくり効果における演算式

表の中のx，yは，図4(c)におけるめくり中心線をy軸，効果のかかる方向をx軸とした直交座標によるものを表している．

	area	ページめくり1（図4(a)）	ページめくり2（図4(b)）
x'=	0	x	
	1	$R\times$rcsin(x/R)	
	2	$R\times\pi-R\times\arcsin(x/R)$	
	3	$R\times\pi-x$	$R\times\pi+R\times\arcsin(x/R)$
y'=	―	y	



(a) ページめくり1　　(b) ページめくり2



(c) 上から見た図　　(d) 横から見た図

図4　ページめくり効果の実現

(a)と(b)にページめくりの効果の例を示す．こうした効果を実現するには(c)と(d)に示す数学的なアプローチをとる．(c)は画面を上から見た図であり，Δxとθ_rを定義している．(d)はめくり方向の真横から見た図（効果のかかる方向をx軸方向にとってみた場合）であり，Rと座標の関係を示している．

られます．しかし，斜めにめくった場合には，下側の見える部分と見えない部分の判別（上側と下側のオーバラップ判別）は簡単ではありません．そこで今回は，

1）上側（area2，3）と下側（area0，1）を別々に処理

2）一度下側の画像をすべて描画した後，上側を描画する

という手順によって，結果的に下側が上側に上書きされて見えなくなる，という方法をとることにします（ただし，上側の変換後の座標がソース画面上に存在することが前提となる）．この方法を採用するには，1画素を2度書きする処理量が必要となりますが，NTSCの場合，1画素の処理レートが13.5MHzであり，今回のターゲット・デバイス（Spartan-3）にとっては余裕のある速度です．2倍の処理量が必要であるとしても，陰面消去の手間に較べれば問題にはならないと判断しました．

● 波効果――波の関数演算以外はページめくりと似ている

図5に，波効果の例と実現方法を示します．制御パラメータとしては，以下の五つが挙げられます．

- 波長：R
- 波高：m_gain
- 位相：$\varDelta x$
- 波の方向：θ_r
- 波の形（正弦波，三角波，矩形波）

図5（d）は画面を上から見た図で，$\varDelta x$とθ_rを定義しています．図5（e）は波の伝わる方向を真横から見た図で，Rと座標の関係を示しています．波の形については単純に関数を選択するだけなので，ここではとくに説明しません．

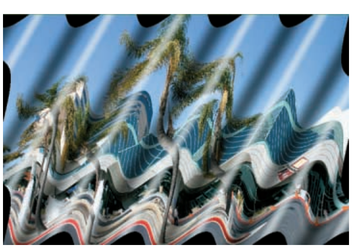
基本的な処理は，波の伝わる方向をx軸にとって波高値を算出し，これをy軸方向に加算すれば波を表現できます．式で表すと，以下のようになります．

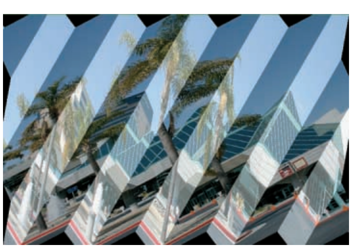
$$x' = x$$
$$y' = y + \mathrm{wave}((x - \varDelta x)/R) \times m_gain \qquad (1)$$

ただし，wave()は波の形を表す関数

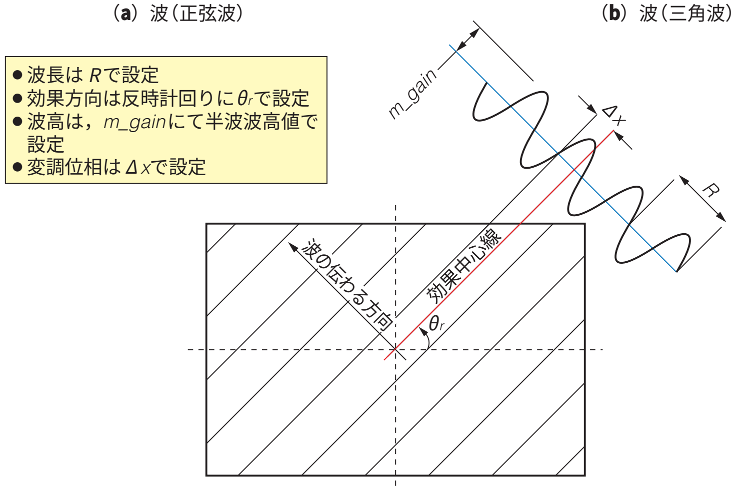
これをさらにθ_r方向に展開してやらなくてはなりません．この手順は，次のとおりです．効果方向に垂直で原点を通る直線を効果の中心線とすると，

- x-y座標より効果中心線からの距離を算出
- 効果中心線からの距離をxとして式（1）におけるwave $((x - \varDelta x)/R) \times m_gain$の項を演算
- その演算結果をx-y成分に分離し，効果中心線を基準に加算

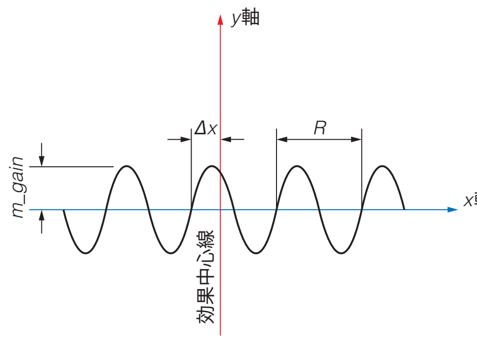

（a）波（正弦波）


（b）波（三角波）


（c）波（矩形波）



（d）上から見た図



（e）横から見た図

図5　波効果の実現
（a）～（c）に波効果の例を示す．こうした効果を実現するには（d），（e）に示すような数学的アプローチをとる．

3

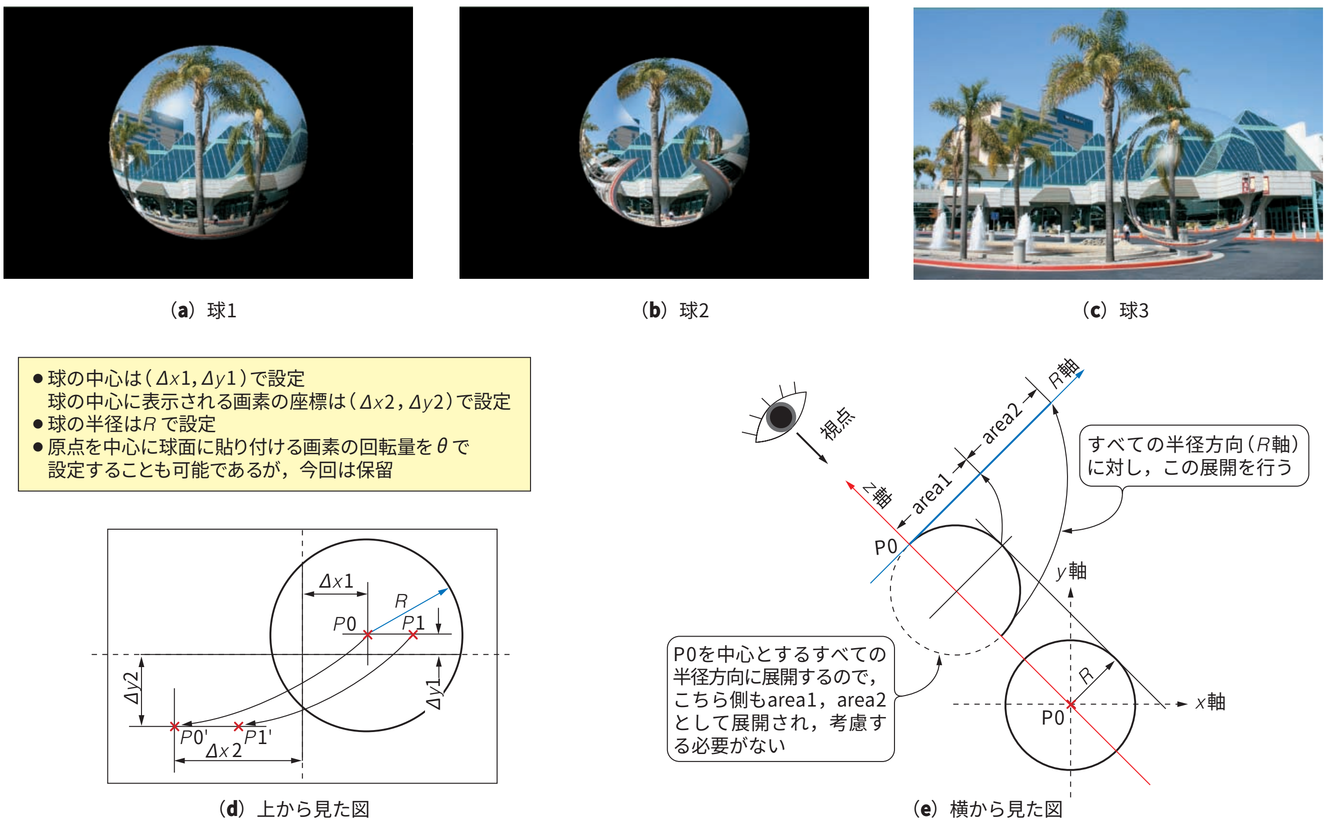


図6　球効果の実現
球効果の実現は，ページめくりの場合の反対側に視点があると考えればよい．

波の関数の演算部以外は，ページめくりで行った操作にきわめて近いことがわかります．なお，この効果では，上側と下側というものはありません．視点からはすべての面が可視だからです．

● 球効果――ページめくりの反対側に視点を置いて考える

最初にお断りしておかなくてはならないのですが，ここで実現する球の表現は，みなさんがふだんテレビなどで見かけるものとは違っているはずです．テレビでよく見かけるのは，世界地図で言うところの円筒図法で表現された地図を球面に張り戻す方式ではないかと思います．円筒図法は地図が長方形なので，同じ長方形であるソース画面をむだなく球面に貼り付けられ，裏側で左右の辺をぴったり合わせられるという利点があります．

これに対して，ここで紹介するのは方位図法によるもので，地図としては円形ですからソース画面との関係に不自然さを感じるかもしれません．しかし，裏側で外周が長さゼロになっていくようすがおもしろいので，あえてこちらを選びました．裏側で起きている現象は**図6(b)**に見ることができます．この効果は裏面(下側)と表面(上側)を入れ替えただけです．**図6(c)**は下側を変形させず，その上に上側の画像を貼り付けてレンズのような効果を得ています．

さて，球の実現方法ですが，実はページめくり効果と密接な関係にあります．**図4(d)**における点P0＝P0 'を表側の中心と考えれば，area0，1が球の半径方向にソース画面を貼り付ける操作になります．つまり，ページめくりの場合の反対側に視点があると考えればよいのです．これでz軸方向の距離を半径方向に展開する処理は説明が終わってしまいますが，次にこの処理をx-y平面上に展開していくという操作が必要になります．それも，ページめくり効果や波効果の場合のような固定の方向ではなく，中心から全方位に展開してやることになるので，極座標的な処理が必要になります．このようすを，**図6(d)**，**(e)**に示します．ここでの制御パラメータは，以下のとおりです．

- 半径：R
- 球の中心位置：(⊿x1，⊿y1)
- 貼り付け位置：(⊿x2，⊿y2)

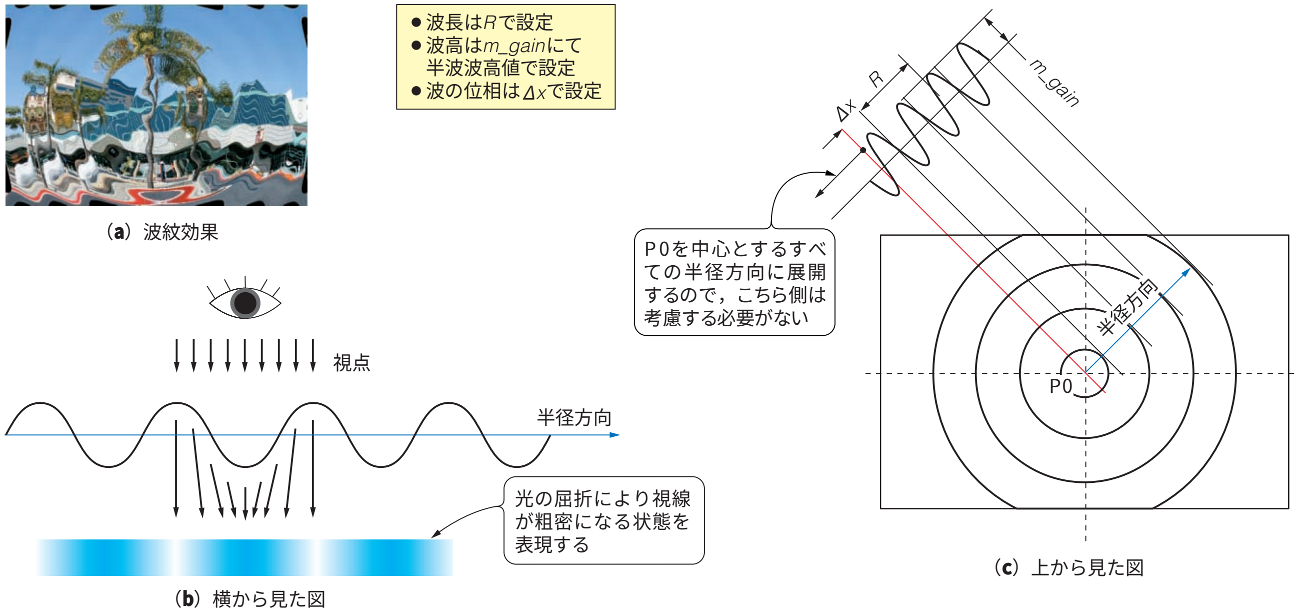


図7　波紋効果の実現
半径方向の座標から生成した正弦波を，半径方向の座標自身に加算することで波紋効果は実現できる．

● 波紋効果——波高値を x 軸方向に加算して実現する

波紋効果は，波効果と球効果の手法の組み合わせで実現できます．波効果において波高値を y 軸方向に加算したのに対し，ここでは x 軸方向に加算することで粗密波を作ります．そのようすを**図7**に示します．その上で，極座標方向に展開することで波紋となるわけです．方法は，球の場合と同じです．制御パラメータは，次の四つです．

- 波長：R
- 波高：m_gain
- 位相：$\varDelta x$
- 波の形（正弦波，三角波，矩形波）

波効果と同様に，波紋効果も全面可視なので，上側と下側の区別はありません．

● 各効果の制御パラメータを一覧表にする

今回筆者が設計した回路では，上記の効果のほかに，**図1**に示したモザイクや砂嵐，引き伸ばしなども実現します．

モザイクは，アドレスをバイナリで考えて，特定のビット以下を丸めてしまうことで実現できます．しかし，これではモザイクの四角の1辺が2のべき乗サイズに限定されてしまっておもしろくありません．ここでは，1辺の長さを連続的に変化させます．すなわち，モザイクの辺の長さを制御するしかけが必要になります．このしかけは，モザイクの1辺が丸められるけた（大きさ）になるように座標をいったん変換したうえで丸め操作を行い，その後で元のサイズに戻すことで実現できます．あとは，その処理を一定方向に展開するだけです．これはページめくりにおける方法と同じです．制御パラメータは，次の二つとなります．

- モザイク1辺長：R
- モザイク方向：θ_r

砂嵐の実現はとても簡単です．1フレームの中で固定の乱数を x-y 座標に加算すると，各画素はあらぬ方向に移動するため，乱数にゲインをかけてやれば連続的に移動量を制御できます．したがって，制御パラメータは，ゲイン m_gain のみとなります．

さて，ここまででだいたいの実現方法は読者のみなさんにも見えてきていると思います．引き伸ばし効果については，みなさんに考えてもらうこととしましょう．

ここまでのまとめの意味で，**表2**にパラメータの一覧表を示します．

3 数式上で似た機能を見つけ出す

ここからは，似ている部分があっても少しずつ異なる各効果の演算を統合する作業に入ります．

前述の内容を見渡して，各効果に必要な処理が，

- 各効果のキャラクタを決める非線形な操作
- 副次的効果を与える効果の方向操作

表2 パラメータ一覧

効果名	R	Δx	θr	m_gain	波形			
					arcsin	正弦波	三角波	矩形波
ページめくり1，2	めくり半径	めくり量	めくり方向	－	○	－	－	－
波1，2，3	波長	位相	波の方向	波高	－	○	○	○
球	半径	位置	－	－	○	－	－	－
波紋	波長	位相	波の方向	波高	－	○	○	○
モザイク	モザイク1辺長	－	モザイク方向	－	－	－	－	－
砂嵐	－	－	－	移動量	－	－	－	－
引き延ばし	－	効果範囲	－	－	－	－	－	－

の二つに大きく分かれていることにお気づきでしょうか．一度なんらかの変換を行ってから，曲げるとか引っ張るとかの操作を施して，元に戻す，といった表現が繰り返し出てきているはずです．

ではまず，これらの操作をどう分類し，どう統合していくかの方向づけを行いましょう．ここが回路化した場合の「構造の簡素化」，「規模の抑制」，「拡張性」を大きく左右するので，じっくりと検討します．

● デバイスの特徴を生かしてテーブル近似を有効利用

ページめくり効果では，arcsin（x/R）という非線形な関数を扱うため，なんらかの関数発生回路が必要となります．今回はこの関数をテーブルによる近似で表現することにします（p.54のコラム「関数のテーブルによる実現」を参照）．タネを明かすと，ターゲット・デバイスであるSpartan-3には，Block RAMや分散RAM注など，テーブルを実現するのにつごうの良い材料がそろっているので，これを用いようということです[(4)]．

1）ページめくり効果の処理手順

ページめくり効果では，幸いなことに，テーブルを引く範囲は$-R \leqq x \leqq R$に限定されています．つまり，$-1 \leqq x/R \leqq 1$となり，実際には$0 \leqq x/R \leqq 1$の範囲がテーブルとしてあればよいことになります．Rのサイズに合わせたすべてのテーブルを持つ必要はありません（まず，物理的に不可能だが…）．ここで行う操作の手順をあらためて見ると，次のようになっています（**手法1：ゲイン調整**）．

- 半径Rを1に固定したテーブルを1種類持つ
- テーブル参照のためのアドレスを半径1に正規化する
- テーブル出力を半径Rサイズに戻す

次に，めくり量はどのように制御すればよいのでしょう．めくり量に合わせたテーブルをすべて持つことは物理的に不可能です．ここではP0＝P0’の不動点を原点に移動してしまえば，テーブルを参照する必要のある最初の点がアドレス＝0となり，それほどめんどうではなくなります（**手法2：原点移動**）．

z方向の情報をテーブルの参照とわずかな演算で実現できるようになったので，今度はめくり効果の方向をどう制御するかを考えます．テーブルを参照するためのアドレスは，めくり方向の原点からの距離になります．したがって，ラスタ・スキャンで与えられたx-y座標をめくり方向の原点からの距離に変換します．これでテーブルを参照すれば，めくり方向の変移量を求められます．この変移量をめくり方向からx，y軸方向にベクトル分割して，その値をラスタ・スキャンで与えられたx-y座標に加えることで，やっと所望の変換が完了します．

ストレートに考えるとこうなるのですが，やはりめんどうな感じがします．また，手法1，2およびほかの効果と共存できるかどうか心配です．

そこで，筆者はこう考えました．原点を中心にx-y座標を回転してめくり方向をx軸にそろえてしまいます．そうすると，x座標のみを考えればよくなります．これでテーブルを参照して変移量をx座標に加え，元の方向に再度回転して戻します（**手法3：方向回転**）．こうすると実にシンプルに考えることができます．ちなみに，やっていることは先のストレートな方法となんら変わることがなく，発想を転換しているだけです．

以上の操作をふまえて，ページめくりに必要な処理手順を図8に示します．これでもシンプルに見えない，とおっしゃる方もいるかと思いますが，このあたりは後ほどまとめて説明します．

以上を基準として，ほかの効果についても考えていきましょう．

2）波効果の処理手順

波効果では，非線形効果として正弦波，三角波，矩形波

注：CLB（Configurable Logic Block）のLUT（look-up table）を用いてメモリを実現する方法．

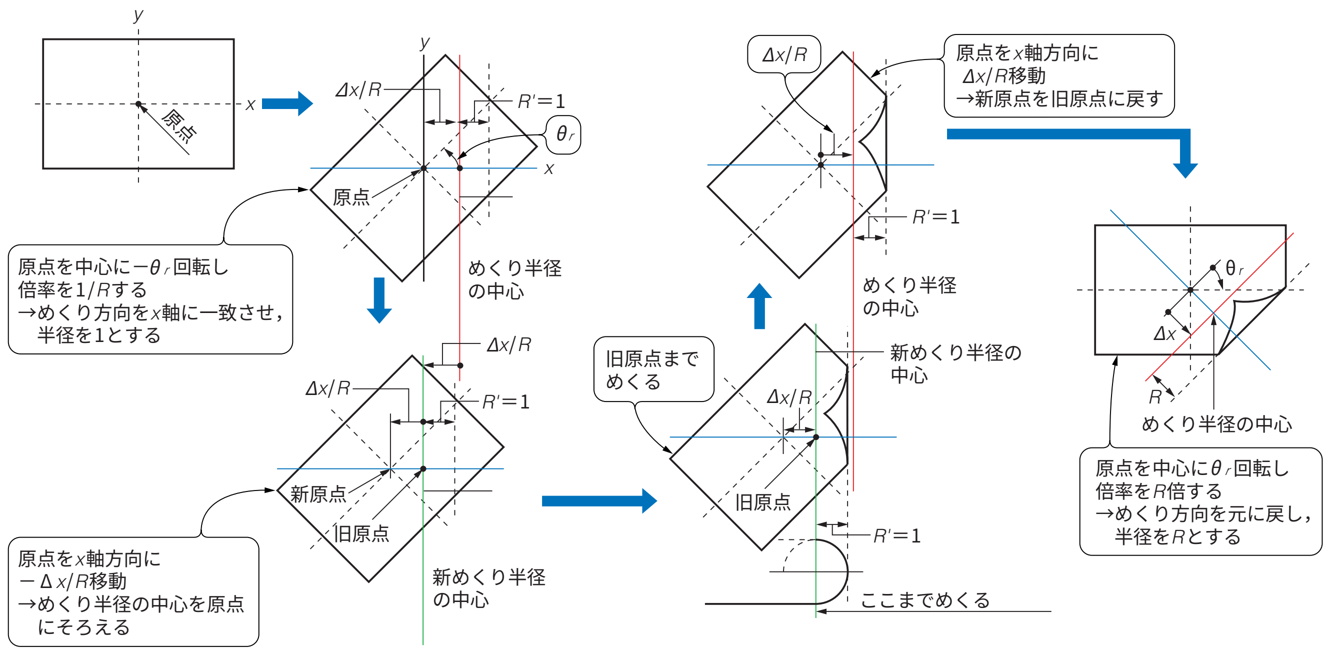


図8　ページめくり処理の手順

まず，原点を中心にx-y座標を回転させて，めくり方向をx軸にそろえる．変移量（Δx/R）をx座標に加えて，めくり半径の中心をそろえる．必要量めくったら元の方向に回転して座標を戻す．

が必要になることはすでに説明したとおりですが，これをテーブルで実現することにします．テーブルとしては，各波のテーブルを波長＝1に正規化した1波分だけ持ち，波高も1としておきます．まず，テーブルを参照するにあたり，前処理線形変換を行います．

- 手法3で，原点周りに逆回転して効果の方向をx軸に合わせる（回転$-\theta_r$）
- 手法1で，x-y座標を$1/R$倍して波長を1に正規化する（$\times 1/R$）
- 手法2で，テーブルの先頭アドレスにあたる部分を原点に合わせる（$-\varDelta x/R$）

ここから非線形変換を行います．

- ここまでで得られたx座標でテーブルを参照する
- 得られた波高値1の波にm_gain/Rをかける
- $1/R$倍したy座標に加算する

これで，非線形変換が終了します．ここからは後処理線形変換を行います．

- 手法2で，元の原点からの距離に戻す（$\varDelta x/R$）
- 手法1で，x，yともにR倍する（$\times R$）
- 手法3で，原点周りに回転させる（回転θ_r）

以上ですべて終了です．

3）球効果の処理手順

球の場合はすでに説明したように，必要となる非線形関数はページめくりと同じです．そのため，新たなテーブルを用意する必要はありません．

一つ大きく異なる点は，ページめくりは一定の方向にめくるのに対して，球は球の中心に向かってめくる操作が必要となる点です．したがって，めくる方向であったθ_rが1フレーム内で定数とはならず，そのときスキャンしているx-y座標によって決まる変数となります．

まず，前処理線形変換を行います．

- 手法2で，球の中心にあたる部分を原点に合わせる（$-\varDelta x1$，$-\varDelta y1$）
- 手法3で，原点周りに逆回転して，すべての点をx軸上に写像する（回転$-\theta_r$）
- 手法1で，x-y座標を$1/R$倍して波長を1に正規化する（$\times 1/R$）

ここから非線形変換を行います．

- ここまでで得られたx座標でテーブルを参照する
- 半径が1の変移量が得られたので，areaに応じて必要な補正演算を行う

これで非線形変換の終了となります．ここからは後処理線形変換を行います．

- 手法1で，x，yともにR倍する（$\times R$）
- 手法3で，原点周りに回転させる（回転θ_r）
- 手法2で，球への貼り付けの中心からの距離を加算する（$\varDelta x2$，$\varDelta y2$）

4）波紋効果の処理手順

波紋の場合もすでに説明したとおり，波と球の組み合わせとなり，必要となる非線形関数は波と同じです．

まず，前処理線形変換を行います．

- 手法3で，原点周りに逆回転して，効果の方向をx軸に合わせる（回転$-\theta_r$）
- 手法1で，x-y座標を$1/R$倍して波長を1に正規化する（$\times 1/R$）
- 手法2で，テーブルの先頭アドレスにあたる部分を原点に合わせる（$-\varDelta x/R$）

ここから非線形変換を行います．

- ここまでで得られたx座標でテーブルを参照する
- 得られた波高値1の波にm_gain/Rをかける
- $1/R$倍したx座標に加算する

これで非線形変換が終了するので，ここからは後処理線形変換を行います．

- 手法2で，元の原点からの距離に戻す（$\varDelta x/R$）
- 手法1で，x，yともにR倍する（$\times R$）
- 手法3で，原点周りに回転させる（回転θ_r）

これ以上説明を続けるとくどくなるので，モザイクや砂嵐，引き伸ばしについては説明を割愛します．

● NTSCではアスペクト比の調整が必要

効果そのものではありませんが，NTSCというフォーマットを使用するにあたって必要な操作があります．それはアスペクト比の調整です．NTSCでは規格上，画素数が720×486と定義されていますが，画面の縦横比（アスペクト比）は4：3とされています．しかし，上記処理における画素数の縦横比は4：3ではありません．つまり，画素が正方形ではないのです．したがって座標が示す格子も正方格子ではないので，これまで説明してきた操作をそのまま適用すると，歪んだ形で表示されます（例えば，球が円ではなくだ円に表示される）．そのため，いったん正方格子に変換してから，これまでの操作を行わなくてはなりません．

● 「必要な要素を一覧にする」ことも重要なスキル

さて，手法3に回転が出てきましたが，座標の回転というと，行列を用いた表現がよく用いられます．実は，手法1の拡大・縮小，手法2の移動も例外ではありません．これらの操作は，すべて行列（線形変換）で表現できるのです．**図8**において回転と半径の倍率変更を同時に行っているのは，これを意識したものです．

ここまでの効果実現において，線形変換で実現できる範囲を**図9**に示します．文章で離散的に記述されたものをただ表などにまとめただけでは，全体像を正確にとらえることは難しいと思います．しかし，**図9**のように一覧の形で

cos，sinはパラメータによって設定される定数

	行列8	行列7
ページめくり1（図4（a）） ページめくり2（図4（b））		$\begin{bmatrix} \cos\theta\times Rx & -\sin\theta\times Rx & 0 \\ \sin\theta\times Ry & \cos\theta\times Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，回転および半径正規化（戻し）
波1（図5（a）） 波2（図5（b）） 波3（図5（c））		$\begin{bmatrix} \cos\theta\times Rx & -\sin\theta\times Rx & 0 \\ \sin\theta\times Ry & \cos\theta\times Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，回転および周期正規化（戻し）
球1（図6（a）） 球2（図6（b）） 球3（図6（c））	$\begin{bmatrix} 1 & 0 & \varDelta x3 \\ 0 & 1 & \varDelta y3 \\ 0 & 0 & 1 \end{bmatrix}$ 貼り付け画像の移動	$\begin{bmatrix} Rx & 0 & 0 \\ 0 & Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，半径正規化（戻し）
波紋（図7（a））		$\begin{bmatrix} Rx & 0 & 0 \\ 0 & Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，周期正規化（戻し）
モザイク，砂嵐（図1）		$\begin{bmatrix} \cos\theta\times Rx & -\sin\theta\times Rx & 0 \\ \sin\theta\times Ry & \cos\theta\times Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，回転および周期正規化（戻し）
引き伸ばし（図1）		$\begin{bmatrix} \cos\theta\times Rx & -\sin\theta\times Rx & 0 \\ \sin\theta\times Ry & \cos\theta\times Ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$ アスペクト変換，回転（戻し）

図9　非線形変換の前後に要求される線形変換

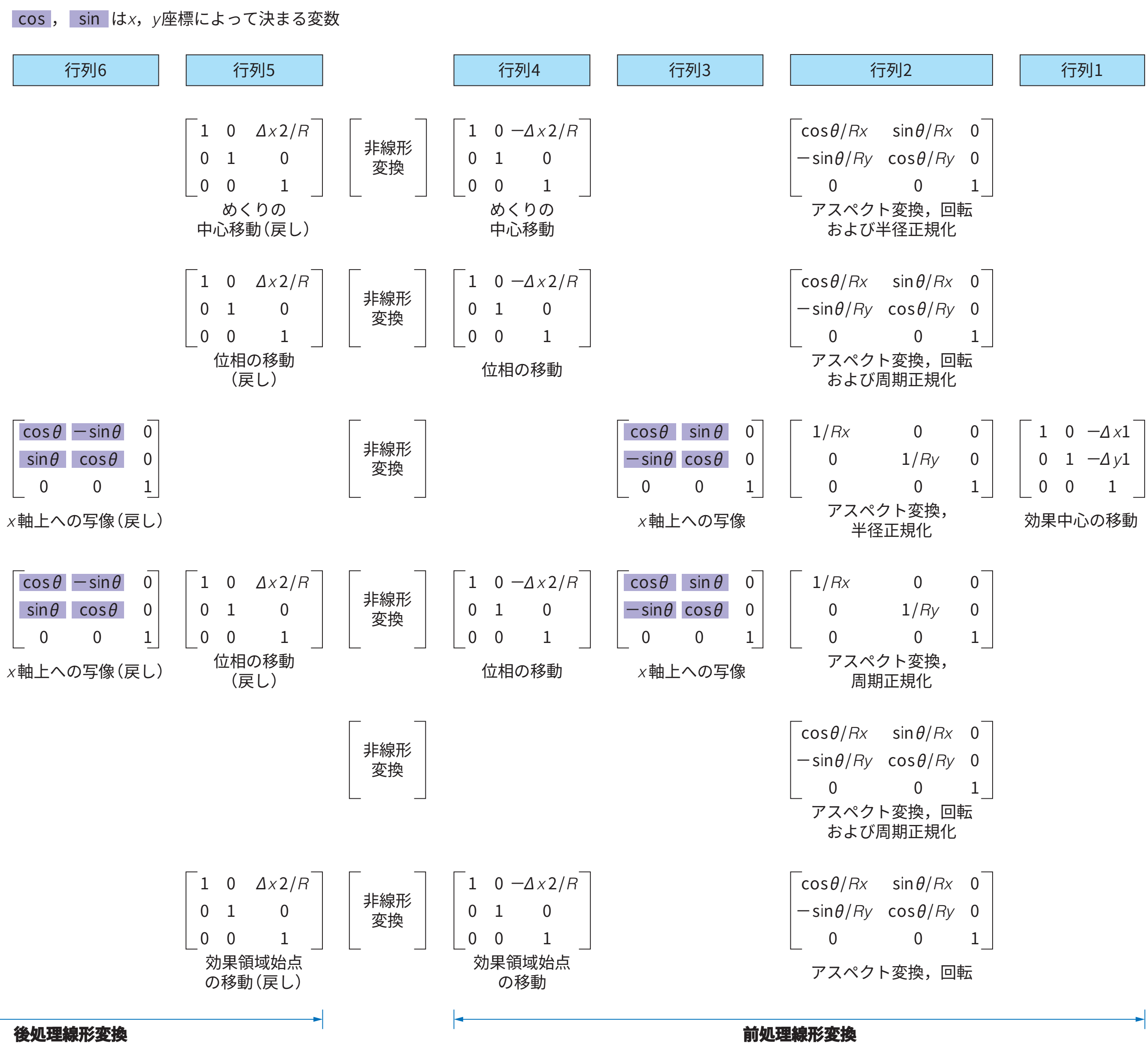
各効果における処理は，行列1から行列8に向かって進められる．図のように，必

必要な要素を見られるようにまとめることは，非常に重要な手法であると筆者は考えています．要素として欠けているものはないか，バランスの悪い部分がないかなどを大局的に把握できるからです．

図9では，非線形変換を挟んで3×3の行列が最大四つ乗算されていますが(空いている部分には単位行列を入れる)，これらはかならずしもハードウェアで演算する必要はありません．フィールド単位でしか変化しないものはソフトウェア側で演算して，一つの3×3行列で与えればよいのです．ただし，球や波紋の効果に見られる画素単位で回転量が変化する部分はハードウェアのしごととなります．そのため，この処理を挟んだ行列処理はソフトウェアで一括というわけにはいきません．

以上より，線形変換部で担当する範囲は，先に説明した副次的効果を与える効果の方向操作，非線形変換への前処理，後処理になります．

次に非線形変換に必要な関数のリストアップを行います．前述したように，非線形変換では各効果の実現を行います．



要な要素を一覧の形でまとめるのは，設計においては重要な手法である．なお，ここでの表記法は，(x，y)の位置ベクトルが列ベクトルで表記される方式を用いている．

3

くどくど説明するのはやめて，一括して**表3**に示します．この表は，線形変換によって正規化された値を扱うことを前提としています．

● トップ・レベルの役割分担を明確にする

これまで長い間説明してきましたが，これでようやくトップ・レベルにおける役割分担が明確になりました．これに基づいてトップ・レベルのブロック図を**図10**に示します．しかし，ここで示すのはあくまで概念図です．

ここまでの作業のポイントは，以下のとおりです．

- **数値演算である限りは実現する数式があるので，まずは数式レベルで最適化を行う**（「なんとなく始められそうなところから手を付ける」とか，「すぐにコーディングを始めてしまう」とか，「そのうちなんとかなるだろう」というのでは，最適な回路実現はおぼつかないと考えよう）
- **何種類もある演算の中で同時に実行する必要のないものどうしは，場合分けして別々に実現するのではなく，類似性を導いて数式上で統合する**（場合分けの多いアルゴリズムは，ハードウェア，ソフトウェアを問わず，くふうが足りないのではないかと一度疑ってみよう）
- **ソフトウェア側で処理できる演算は，とことんハードウェアに持ち込まないようにする**（必要な演算が明確に分類されていないと，きちんと線を引くことはできない）
- **時には発想を転換して，スマートな実現を心がける**（なんでもかんでも，ストレートに表現すればよいというものではない）

4 トップ・レベルの構造を考える

ここからやっとハードウェアらしい話になってきます．

表3
非線形変換に必要な関数一覧
表には示していないが，球2は球1の上側と下側を入れ替えることで実現できる．

効　果		area0 ($x<-1$)	area1 ($-1 \leqq x<0$)	area2 ($0 \leqq x<1$)	area3 ($1 \leqq x$)
ページめくり1	上側	$-x+\pi$		$\pi-\arcsin(\mathrm{abs}(x))$	nop
	下側	x		$0+\arcsin(\mathrm{abs}(x))$	nop
ページめくり2	上側	nop	$\pi+\arcsin(\mathrm{abs}(x))$	$\pi-\arcsin(\mathrm{abs}(x))$	nop
	下側	x		$0+\arcsin(\mathrm{abs}(x))$	nop
球1	上側	nop		$0+\arcsin(\mathrm{abs}(x))$	nop
	下側	nop		$\pi-\arcsin(\mathrm{abs}(x))$	nop
球3	上側	nop		$0+\arcsin(\mathrm{abs}(x))$	nop
	下側	x			
波		$y+gain \times \mathrm{wave}(x)$			
波紋		nop		$x+gain \times \mathrm{wave}(x)$	
引き伸ばし		x		$=0$	
モザイク		$\mathrm{floor}(x)+0.5$　$\mathrm{floor}(y)+0.5$			
砂嵐		$x+gain \times x\ random$　$y+gain \times y\ random$			

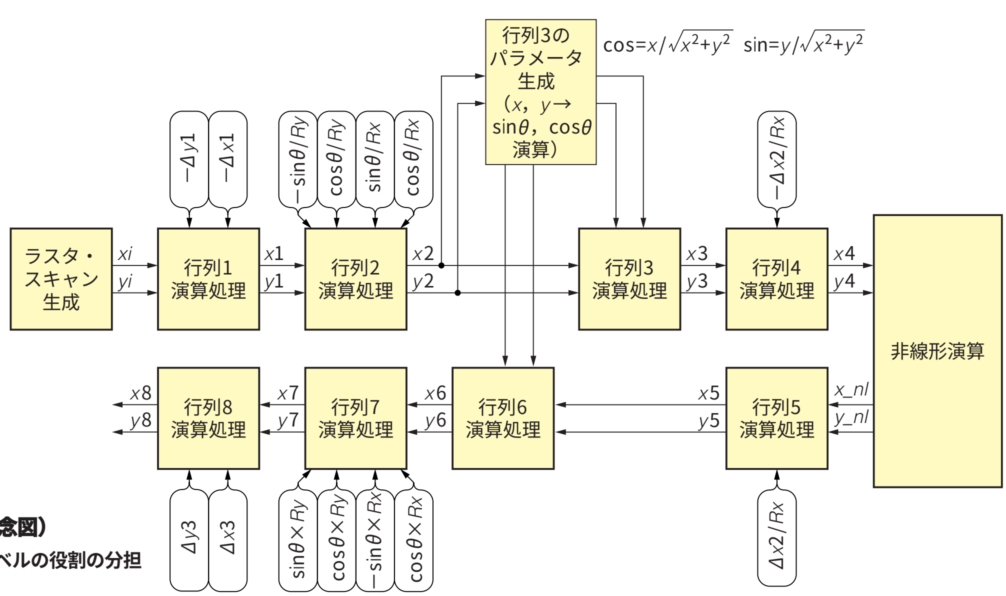


図10　アドレス生成ブロック図（概念図）
ここまでの作業によって，トップ・レベルの役割分担が明確になる．

● 似ている演算に着目して乗算器を減らそう

図9の行列演算を右辺に$[xi, \ yi, \ 1]^T$（^{T}は転置行列を表す）を置いて順次処理した場合，行列が表示されていない部分に単位行列をあてると，非線形変換前だけで8回の乗算を必要とします．ここではシンプルで小さな構造を目指しているので，もう少しくふうしてみましょう．

非線形変換前で乗算が必要となるのは，行列2，3においてですが，行列3は球と波紋でしか使用しません．そして，そのとき行列2に割り当てられた機能は，アスペクト比の変換と半径（周期）の正規化だけです．乗算は$1/Rx \times xi$と$1/Ry \times yi$の2回だけであり，加算は必要としません（そのほかの効果においては，4回の乗算と2回の加算が必要になる）．よって，そのほかの効果における行列2の機能を行列3に割り当ててしまえば，行列2としては2回の乗算のみが残るので，非線形変換前に必要な乗算を6回に減らすことができます．これは，非線形変換後も同じように適用できます．

演算の内容ではなく，演算の類似性に着目して割り当てを変え，演算量を減らしてみました．

● 内部クロックの高速化によって乗算器を減らそう

前述したように，効果によっては上側の描画と下側の描画を行わなければなりません．上下の重なりぐあいを考えずに処理を行うには，1フレーム内で2フレーム分描画できる処理能力が必要となります．NTSCの画素クロックは13.5MHzなので，最低でも27MHzレートの処理が必要なわけです．

ところで，ターゲットであるSpartan-3の実力から考えると，8倍レートの108MHzなら十分守備範囲といえます．内部で使用するクロックを108MHzとすれば，27MHzの4倍の処理が行えます．つまり，次の式における乗算は1個の乗算器に割り当てることができます．

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ d & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a \times x + b \times y \\ d \times x + e \times y \\ 1 \end{bmatrix} \quad (2)$$

これによって実際の乗算器の数を大幅に減らすことができます．なお，式(2)は図9における1回分の行列演算を抽出したものです．

● 要求仕様に応じて演算方式を選択しよう

1）浮動小数点演算の使いどころ

非線形変換の両側にある線形変換回路では，回転や縮尺の変換などを行っていますが，いずれも主たる効果を生むものではなく，非線形変換をスムーズに行うのが目的です．したがって，前半の処理と後半の処理を行った結果，最後には元に戻ることが大前提となります．このため，精度が悪いと各画素の連続性が悪くなり，非常に汚い画像になってしまいます．

また，ページめくり効果におけるめくられていない部分のような，変形の加わっていない部分は，できる限り入力と同じものが出力されなくてはなりません．例えば，業務用の機器などの場合は完全にそれを保証する必要があります．今回は業務用のレベルを目標としてはいませんが，高い精度が必要であることには変わりありません．

さらに，半径の正規化などで縮尺を大幅に変更する機能が必要となります．例えば，球の効果を使う場合，変形していない最初の画面から緩やかに変形し，球からやがて見えないくらいの点になって消える，といったことも必要になります．そのため，最初と最後の半径の差は，数十ビットになります．しかし，最終的に必要な精度は，変換前の座標が持っていた精度にすぎません．このようなアプリケーションには，浮動小数点演算が最適です．ここでは，基本的なことが理解されているという前提で話を進めます（浮動小数点演算についての説明は割愛する．本特集 第2章や参考文献(2)を参照のこと）．

浮動小数点演算というと，回路規模が大きいことで敬遠する方が多いのですが，いつもIEEE 754の単精度や倍精度の規格どおりでなければならないというものではありません．例えば，IEEE 754では単精度でも指数部は255けた取ることができますが，今回のアプリケーションでは明らかにすべてのけたが必要とはなりません．必要なけたの部分のみ使用すれば，意外に回路規模は爆発しないものです．また，外部のアプリケーションとデータを共有しない条件の下では，規格どおりのフォーマットである必要はないともいえます．

では，どの程度の精度があればよいかですが，NTSCの全画面を座標で表現するのに符号付きで整数10ビットが必要になります（原点を画面中央とするので符号付きになる）．また，変形の連続性をスムーズに表現するためには，小数点以下の値が必要ですが，業務用でなければ小数点以下4ビ

ットあれば十分なので，ここでは4ビットに設定します．合計14ビット精度が必要となりますが，ここでは仮数部を17ビットで扱うことにします．3ビット余分ですが，Spartan-3のハードウェア乗算器の入力が符号付き18ビット幅であるため，こうしたほうが程よく収まります．けた数は±32けたあれば問題ないと判断し，指数部は符号付き6ビットとします．

2）固定小数点演算の使いどころ

乗算，除算については浮動小数点演算が便利なのですが，加算や減算については固定小数点演算の場合と比べて明らかに規模が大きくなります．今回のアプリケーションでは，行列1，4におけるオフセット加算，および行列2，3における乗算後の加算を固定小数点で行います．ラスタ・スキャンによる座標入力は固定小数点ですし，非線形演算の入出力は固定小数点となるので（詳細は後述），固定小数点で演算したほうが規模への影響が少ないと判断しました．

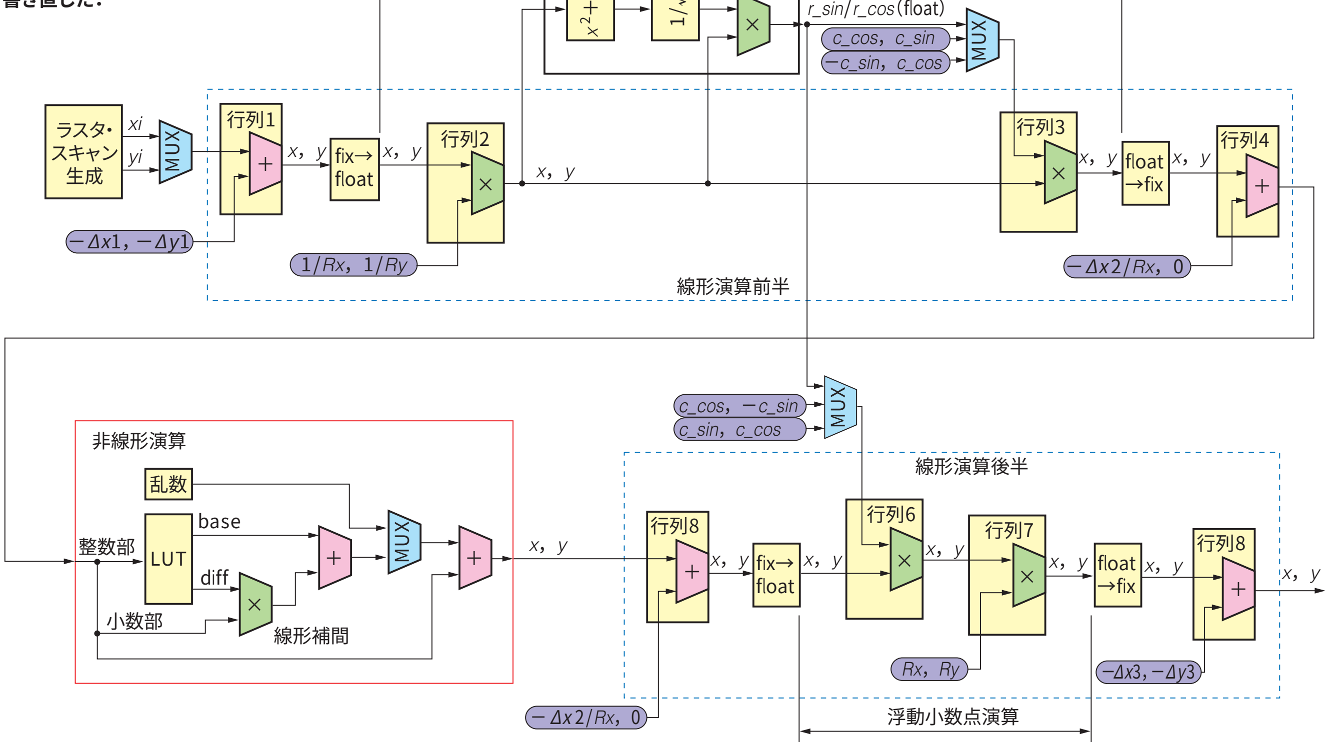
一方，非線形演算は変形が目的で，連続性を保って変形すれば，それほど精度を要しないというのが前提です．例えば，球面がギクシャク（でこぼこ）せずに表現されれば，真球である必要はないということです．計測回路を設計しているわけではないので，人の目に球と映れば，それ以上の精度は必要ありません．よって，非線形演算も固定小数点で行うこととします．

3）テーブルによる関数の発生

演算する関数としては，arcsin，正弦波，三角波，矩形波などが必要です．すでに述べたようにSpartan-3のリソースのつごうで，表現しにくい関数はテーブルによって発生させることにします．波に使用される正弦波などは，見た目がそう見えればよいので，2次関数を代用してもかまわないのですが（見た目が重要であれば，正弦波の代用としての2次関数は十分実用になる），テーブルのサイズとしては大きいものを必要としないので，統一的な方法で複数の関数を実現したほうが有効と考えました．

これらの検討をもとに，トップ・レベルのブロック図を書き直したものが，**図11**になります．

図11
アドレス生成ブロック図
乗算器を減らすくふうを施し，演算方式を決めて，トップ・レベルのブロック図を書き直した．



5 極座標変換サブモジュールを構築する

極座標変換という表現でここまで説明しましたが，線形変換で回転写像を得るために必要なものは，角度ではなく，cos θ，sin θの値です．まずはここから説明を始めましょう．

ここで必要とされるのは，原点を中心とし，x軸を起点として反時計回りに回転した角度で定まるcos θ，sin θの値とします．さらに，x-y平面は，右上方向をインクリメント方向とすることが多いので，これに合わせることにします．すると，ラスタ・スキャンは，水平方向には左から右にスキャンするのでこれに合致しますが，垂直方向は上から下にスキャンするため合致しません．したがって，垂直方向にはデクリメントで上から下にスキャンすることで，x-y平面に合わせることとします．これにはルールはないので，どのような基準でもかまいません．筆者にとってそれがわかりやすいというだけです．以上の取り決めを，図12に示します．

上記の取り決めに基づいて，cos θとsin θを$(x,\ y)$を用いて表現すると，式(3)となります．

$$\cos\theta = x / \sqrt{x^2 + y^2}$$
$$\sin\theta = y / \sqrt{x^2 + y^2} \quad (3)$$

これを求めるブロック図を図13に示します．大前提としているのは，トップ・レベルを構成した段階で，ここを浮動小数点演算で処理することです．

● 乗算器をじょうずに使い回す

4クロックで1アドレスを生成するのは全体を通して変わらないので，このブロックでもこれに基づいて構成しています．まず入力段の乗算器で行う演算は，xとyを2乗することなので，4クロックのうち2クロックまで使用することになります．残りの2クロックは空き時間になるため，これの有効活用を考えます．

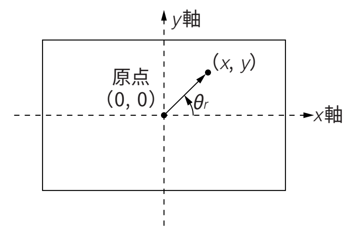


図12 x-y平面と回転方向
x-y平面では，右上方向をインクリメント方向とする．

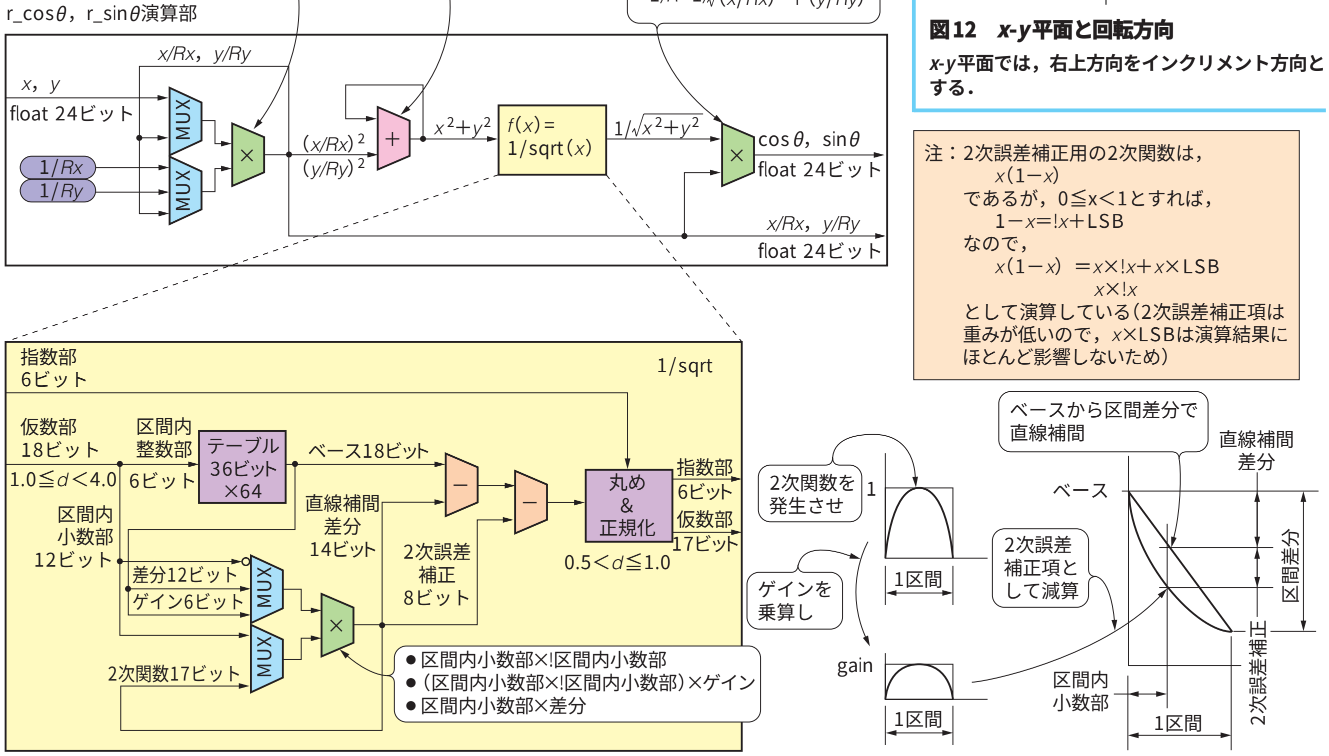


図13 cosθ，sinθ演算ブロック図
このブロックは，浮動小数点で演算することが前提．1/√（平方根の逆数）の演算は，テーブルを使って実現している．

ここのブロックではほかにも乗算器が必要なのですが，1/√（平方根の逆数）は独立した関数ととらえたいところです．また，その後段にあたる分母と分子の乗算はやや離れた演算の印象があるので（観念的な表現で申しわけないが…），それらの機能をこの乗算器に押し込むのは避けたい気がします．

ところで，このブロックが必要なのは球効果と波紋効果の場合に限られます．トップ・レベルで考えると，前段において行列2の機能である$1/Rx$と$1/Ry$の乗算が，アスペクト比の変換や半径正規化として必要になるのも球効果と波紋効果の場合だけです（ほかの効果の場合，行列3としてアスペクト比の変換，半径正規化を行うことにしたため）．したがって，行列演算ではありますが，$x \times (1/Rx)$，$y \times (1/Ry)$の二つの乗算を2乗演算の乗算器で実現することにしました．$1/Rx$，$1/Ry$の乗算を行ってからこれをフィードバックし，2乗することになります．

次段の加算器はxとyの2乗項を加算します．浮動小数点における加減算は，けたをそろえてから行わなくてはならないため，固定小数点の場合に比べて回路規模が大きくなります．できる限り避けたいところなのですが，$1/Rx$と$1/Ry$の乗算を前段で行わなくてはならない関係上，ここは我慢することにしました．お気づきかもしれませんが，ここ以外の加減算はすべて固定小数点で行っています．

● 1/√の関数テーブルはSpartan-3の分散RAMで十分

3段目は1/√の演算を行います．平方根を求めてから逆数を演算するという方法もありますが，関数をテーブルで実現することに決めているので，一括して関数を実現する方法を採用しました．テーブルであれば，いかなる非線形関数でも実現できます．1/√の実現方法についての詳細は，誌面のつごう上，省略します（参考文献（3）を参照するとよい）．ここではポイントのみを説明しましょう．

まず，1/√という関数を浮動小数点演算で実行する場合，指数部が1/2になって符号が反転しますが，指数部は整数なので2で割り切れないことが不つごうになります．正規化された浮動小数点表現といえば一般に$1 \leqq$仮数部< 2なのですが，ここでは入力において$1 \leqq$仮数部< 4とし（整数部が2ビット），指数部を偶数に正規化します．これにより，出力の指数部は整数，$0.5 <$仮数部$\leqq 1$を実現できます．このように1/√の実現においては，浮動小数点演算を採用すると，指数部が2単位で同一の仮数部が出現するという周期性を利用することができ，広い範囲を少ないテーブルで実現できます．

また，$x/\sqrt{x^2+y^2}$のような関数を実現する場合，分母と分子を単独で見ると，非常に広い範囲をカバーする必要があります．しかし，分母と分子は独立に値をとるわけではなく特定の関係があるので，とりうる値は限られた範囲にあります．求めるものが$\cos\theta$なので，これは明らかです．このような場合，限られたけたの範囲で精度を失うことなく求められるのも，浮動小数点演算の恩恵といえます．

次にテーブルのワード・サイズですが，$1 \leqq$仮数部< 4の範囲をテーブルに収めればよいので，この範囲を48区間に均等に割ってテーブルに収めることにします．すでに述べたように，ここでの演算は精度が必要です．精度を上げる手っ取り早い方法は，テーブルを細かく持つことなのですが，効果を上げようとすると指数関数的にテーブル・サイズが大きくなってしまいます．そこで，演算で2次誤差補正まで適用して精度を上げ，テーブルのサイズは小さくすることにしました．この大きさであれば，Spartan-3のBlock RAMを投入するまでもなく，分散RAMで実現できるため効率が良く，配置上の制約もなくて便利です．えたいの知れない（？）関数をたくさん作るときには重宝しています．

テーブル上の1ワードは，ベース18ビット，直線補間用差分12ビット，2次誤差補正ゲイン6ビットの合計36ビットとします．ここでベースとは，48分割された区間の先頭における関数の値（出力値）とします．また，直線補間用差分は，次の区間の先頭値までの差分です．直線補間というと，$(1-k) \times a + k \times b$の形を想像される方も多いでしょうが，$a + k \times b$の形に持ち込めるのであれば，こちらのほうが演算としては簡素ですし，多くの場合，差分のほうがベースよりはビット幅が少なくてすむのでテーブルが小さくなります．

ここでの入力の仮数部は，整数2ビットを含む18ビットとします．テーブルを48分割したので，入力仮数部の上位6ビットでテーブルを参照し，残りの12ビットをテーブルからの差分に乗算し，ベースに加算します．これで直線補間は完了です．

2次誤差補正用の関数は$x \times (1-x)$の演算で生成します（xは仮数部の下位12ビット）．このxは48分割された区間の1区間内の位置を表すので，$0 \leqq x < 1$と等価です．そのとき，

$$1 - x = !x + \mathrm{LSB}$$

（!xはxの論理反転，LSBはxの1LSBを指す）

が成立し，次の式のようになります．

$$x \times (1 - x) = x \times !x + x \times \mathrm{LSB} \quad x \times !x$$

2次誤差補正項そのものの値が小さいので，$x \times$ LSBの項は最終結果にほとんど寄与せず，省略することができます．これによりさらに演算を簡略化しています．これに2次誤差補正用のゲイン（区間ごとに異なる）を乗算し，2次補正項を直線補間値に加算して演算が終わり，あとは丸めと正規化を施して完了となります．

ここまでで乗算は3回出てきたので，1個の乗算器で足りることになります．テーブルが簡素なので精度面で支障がありそうですが，17ビット以上の精度は出るため，今回はこれで十分です．

以上で$1/\sqrt{x^2+y^2}$が生成されました．これに分子であるx，yを乗算すれば，cosθ，sinθの完成です．終段の乗算器には，二つの乗算しか割り当てられていませんが，ほかに割り当てるものもないので，ここは良しとしましょう．総合的な精度は16ビット程度です．

6 線形変換サブモジュールを構築する

線形変換回路のブロック図を**図14**に示します．非線形演算ブロックを挟んで前半と後半に分かれます．

このブロックにおける大前提は，整数部10ビット，小数部4ビットの精度を確保するということです．浮動小数点演算を採用するという前提で話を進めていますが，最終的なアウトプットは固定小数点の座標であるということを忘れてはなりません．

● 非線形変換前後の線形処理を実現する

まず，前半部から説明しましょう．入力は4クロック単位で変化するx-y座標です．ラスタ・スキャンでインクリメントすることを想定していますが，アドレス変換系とし

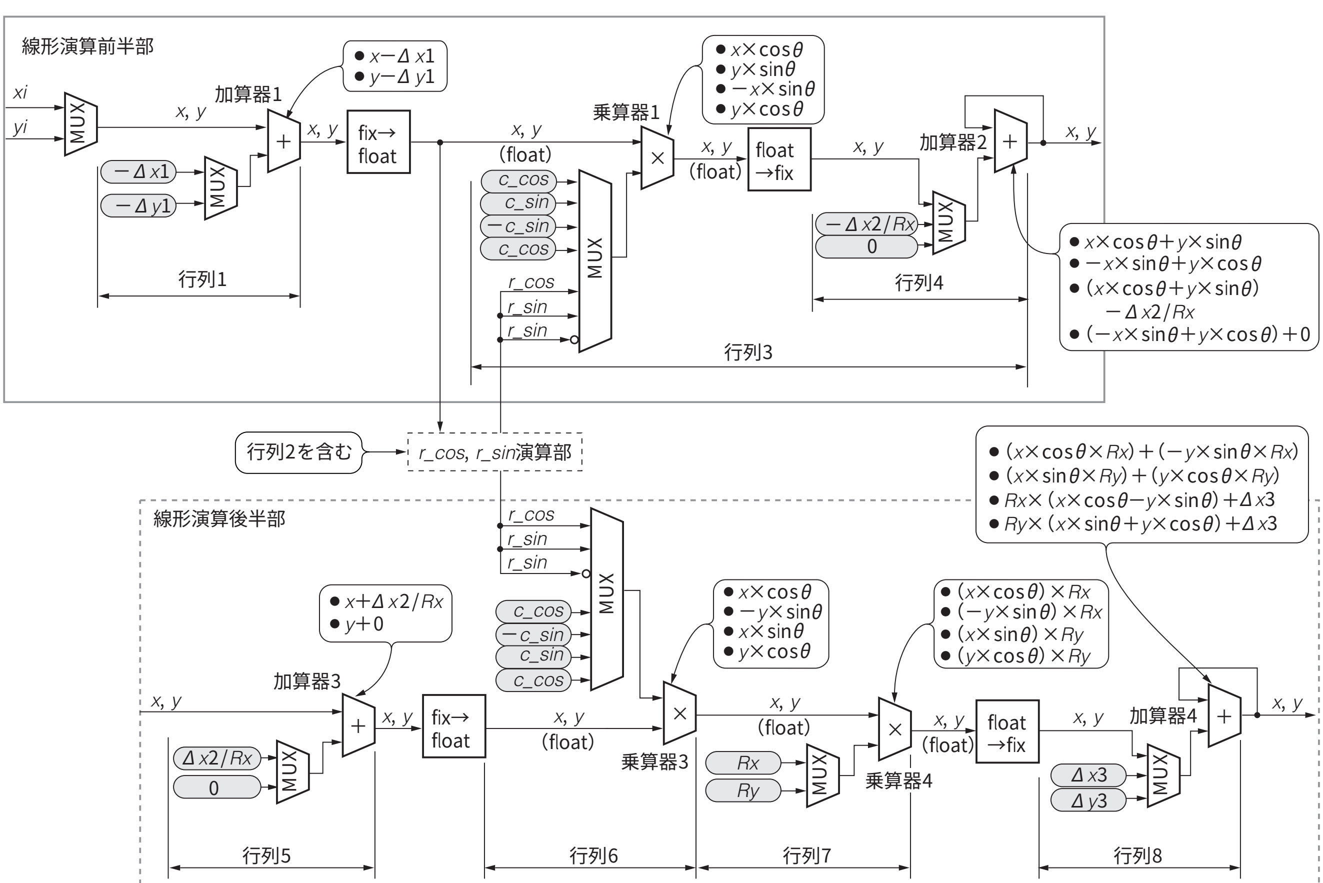


図14　線形変換ブロック図
線形変換ブロックは，非線形演算ブロックを挟んで前半と後半に分かれる．

3

表4　線形変換部における演算

	演算器	フェーズ	演算内容	分　類
前半部	加算器1	0	$x' = x - \Delta x1$	行列1
		1	$y' = y - \Delta y1$	行列1
		2	nop	—
		3	nop	—
	乗算器1	0	$x_x' = x \times (\cos\theta / Rx)$	行列3
		1	$y_x' = y \times (\sin\theta / Rx)$	行列3
		2	$x_y' = x \times (-\sin\theta / Ry)$	行列3
		3	$y_y' = y \times (\cos\theta / Ry)$	行列3
	加算器2	0	$x' = x_x' + y_x'$	行列3
		1	$y' = x_y' + y_y'$	行列3
		2	$x' = x - \Delta x2 / Rx$	行列4
		3	nop	—
後半部	加算器3	0	$x' = x + \Delta x2 / Rx$	行列5
		1	nop	—
		2	nop	—
		3	nop	—
	乗算器3	0	$x_x' = x \times (\cos\theta \times Rx)$	行列6
		1	$y_x' = y \times (-\sin\theta \times Rx)$	行列6
		2	$x_y' = x \times (\sin\theta \times Ry)$	行列6
		3	$y_y' = y \times (\cos\theta \times Ry)$	行列6
	乗算器4	0	$x_x' = x_x \times Rx$	行列7
		1	$y_x' = y_x \times Rx$	行列7
		2	$x_y' = x_y \times Ry$	行列7
		3	$y_y' = y_y \times Ry$	行列7
	加算器4	0	$x' = x_x' + y_x'$	行列6
		1	$y' = x_y' + y_y'$	行列6
		2	$x' = x + \Delta x3$	行列8
		3	$y' = y + \Delta y3$	行列8

てはとくにそれでなくてもかまいません．**図12**の座標系を用いると述べましたが，次段でオフセットの加算を行えるので，そこでの補正を前提とすれば，原点を画面中央とした符号付き座標でなく，左下を原点とした符号なし座標でもかまいません．複数のビデオ・フォーマットに対応する場合，そのほうがシンプルでしょう．その座標を初段でマルチプレクスして，演算器を共用する準備を行います．後段の演算にとってつごうが良いように，時間軸方向にxとyを割り当てます．

ここから行列1～4の演算が始まります．**表4**に，後半部を含めた演算の内容を示します．

まず，行列1の処理であるオフセット加算を加算器1で行います．ここまでは固定小数点による演算になります．次の段では浮動小数点の演算となるので，ここで独自フォーマットの浮動小数点に変換します．

浮動小数点に変換された座標は，行列2の処理を含むcos/sin演算部と，行列3の処理に送られます．球や波紋効果の場合はcos/sin演算部から出力されたパラメータ，それ以外の効果の場合はソフトウェアで設定されたパラメータによって行列3の演算が行われます．乗算器1では行列3の乗算が実行されます．

残る線形変換前半部での演算は，行列3の乗算済みの項の加算と行列4の加算のみとなりますので，ここで固定小数点に戻すことにします．固定小数点に戻した後，上記の加算を加算器2で行います．

これで線形変換前半部の処理を終了し，非線形変換部に固定小数点の座標を渡します．

次に，後半部ですが，ここは前半部と反対の処理手順となります．

非線形変換部から渡される座標に，行列5の処理としての加算を加算器3で行います．加算器3では一つの加算しか行わないのでもったいない気もしますが，ほかに行うものもないのでこれで良しとしましょう．

ここまでが固定小数点での演算になります．次の段では行列6，7の浮動小数点の演算となるので，ここで独自フォーマットの浮動小数点に変換します．

浮動小数点に変換された座標は，行列6の処理に送られます．球や波紋効果の場合，cos/sin演算部から出力されたパラメータ，それ以外の効果の場合はソフトウェアで設定されたパラメータによって行列6の演算が行われます．演算のうち，乗算器3では行列6の浮動小数点乗算が行われます．

球や波紋効果の場合，$Rx = 1$，$Ry = 1$となり，アスペクト調整や半径の戻しは行列7で行います．本来であれば，次に行列6の加算処理を行うところですが，浮動小数点に変換された状態で行列7まで処理を行いたいので，先に行列7の乗算を行います．行列7は行列6の加算が終わっていないので2回の乗算で済むはずですが，4項別々の乗算を行うので，乗算処理は4回になります．これを乗算器4で行います．

残る線形変換後半部の演算は，行列6でやり残した加算と行列8の加算のみとなるので，ここで固定小数点に戻すことにします．固定小数点に戻した後，加算器4で上記の加算を行います．

これで線形変換後半部の処理を終了し，アドレス生成の出力である固定小数点の座標を得ます．

7　非線形変換サブモジュールを構築する

最後に非線形演算ブロックについて説明します．非線形

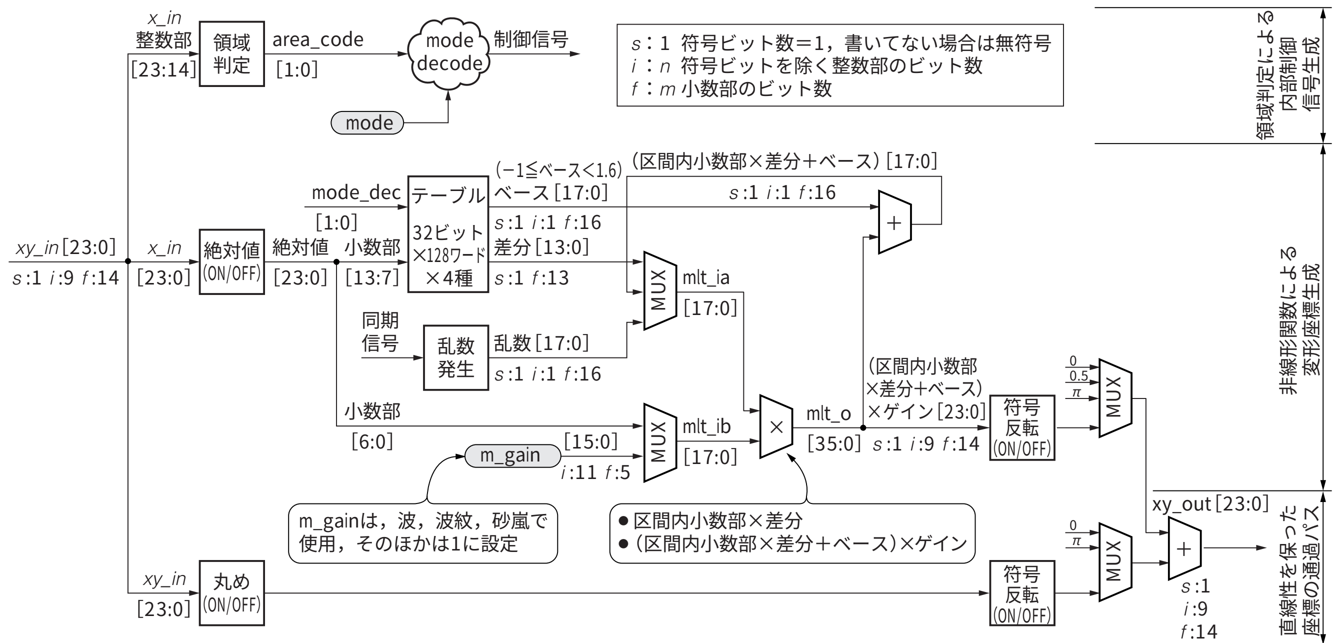


図15　非線形変換ブロック
図から，このモジュールが上段，中段，下段の三つに分かれていることがわかる．

演算ブロックで行う演算は，**表3**に示したとおりです．ブロック図を**図15**に示します．この図では，このモジュールが上段，中段，下段の三つに分かれていることがわかります．ここでの大前提は，「見かけ重視で精度不要」です．

● 上段で領域を判定し，内部制御信号を発生させる

入力の段階で線形変換前半部により，変形のスケールがx軸上に写像され，さらにテーブルを参照するためにスケールが正規化されています．そのため，x座標の値が，－1，0，＋1を境として四つの領域に分離され，効果の内容と各領域によって決まる演算を行うことになります．**図15**の上段は，その領域判定と，効果および領域別の内部制御信号を発生します．主な制御は以下のとおりです．

- 乗算器，加算器の入力制御
- 絶対値機能のON/OFF
- 丸め機能のON/OFF
- 符号反転機能のON/OFF

● 効果のための変形成分をテーブルや乱数で生成

図15の中段は，主たる効果の変形座標を生成する部分になります．ここが要の部分なのでじっくり説明しましょう．

入力段の絶対値回路は，ページめくりにおいて$x=0$を境に折り返すという操作の際に使用します．そのほかの効果では使用しません(機能OFF)．球や波紋でも中心から折り返しているように感じますが，この二つの効果の場合，x軸の正側に座標がすべて写像されているため，負側が存在しないことになり，折り返しは必要ありません．

次段には，メインとなるテーブルがあります．**表3**から明らかですが，ここで必要な関数は，arcsin，正弦波，三角波，矩形波の4種類です．このテーブルにはBlock RAMを使用することにしました．関数の近似はcos/sin演算回路で用いた方法と同じですが，精度を追求しないことにしているので，2次誤差補正までは適用せず，直線補間までとしています．したがって，テーブルの1ワードは，ベース18ビット，直線補間差分14ビット(いずれも符号付き)の合計32ビットとなります．テーブルのワード数は，直線補間を適用するため64ワードもあれば十分なのですが，そうすると一つのBlock RAMの半分で済んでしまいます．もったいないというだけの理由で128ワードを各関数に割り当てています．

また，三角波，矩形波については，わざわざテーブルを用いるまでもないのですが，ほかの関数と処理をそろえるということで，あえてテーブル化しました．ただし，三角波，矩形波には不連続点があるので，区間の区切りをこの不連続点に合わせておかないと，そこでの補間がなまった波形を生成するため注意が必要です．テーブル上で正弦波

関数のテーブルによる実現

今回のディジタル・ビデオ・エフェクトの演算回路では，関数をテーブルで表現する手段を多用しているので，ここで若干の補足をします．

「テーブルによる関数実現は，精度が悪くて使えない」という意見を耳にしたことがあります．どういう事情でそのような判断となったのかは定かではありませんが，現在ではメモリのビット当たりの単価は10年前とは比べ物にならないくらい安くなっています．つまり，テーブルで表現できる範囲は着実に広くなっているということを，まずは認識しなくてはなりません．また，FPGAに搭載されているメモリの量も相当増えてきているので(例えば，今回のターゲット・デバイスであるSpartan-3 XC3S400は56Kビットの分散RAMと288KビットのBlock RAMを備えている)，テーブルの使用について非常に高い自由度が確保されたといえます．なにしろ，すでにチップ上にあるものなのですから，使わないで放っておくのは資源のむだです．

加算や乗算で表現できる関数であれば，多くの場合，直接計算したほうが適当であるということには賛同していただけると思います．しかし，今回使用したような初等超越関数となると，なんらかの近似が必要になります．例えば，入出力ともに8ビットであれば，テーブルとして必要なビットは$2^8 \times 8 = 2{,}048$ビットなので，テーブルのみで実現しても今どきは問題とされないでしょう．しかし，入出力ともに16ビットともなると，1Mビットのテーブルが必要となり，全部テーブルに押し込めるというのは非現実的です．そこで，「必要な精度に応じて，誤差の補正次数を変えて対応する」近似方式を今回は用います[3]．

テーブルの持っている最大の利点は，「いかなる関数も表現できる」ということに尽きます．不連続な関数でもかまいませんし，本文で用いた複合関数も適用可能です．欠点は，うまく実現方法を選択しないと，膨大なメモリが必要になる場合が往々にしてあるということでしょうか．1/√(平方根の逆数)を例にとれば，今回のアプリケーションで必要とされるすべての定義域をテーブルでカバーしようとすると，いったいどれくらいのメモリがあればよいのか想像もつきません．2けた単位で指数部が変わるだけの周期関数として表現したからこそ，現実味があるわけです．

テーブルによる手法は，かなり古典的なものですが，今だから有効に使うことができるということ，また，テーブルという手法の1面だけをとらえて単純に使用するのではなく，なんらかの手段と組み合わせることで効率的に使えるということを覚えておきましょう．

や三角波，矩形波の値域は$-1 \leqq \mathrm{out} \leqq 1$となりますが，arcsinは定義域が$0 \leqq x/R \leqq 1$となり，値域は$0 \leqq \mathrm{out} < 1.6$となります(ここで，outはテーブル出力を意味する)．

テーブルの下には，砂嵐で使用される疑似乱数の発生器があります．ここには20次のLFSR(linear feedback shift register)を用いてx，y両方の乱数を生成し，1フレームごとにリセットをかけて，同一の座標に対して同一の乱数が毎フレーム得られるようにしています．これによって，各画素が連続的に飛散することができます．ほんとうの乱数ではなく疑似乱数だからできることです．

このようにして得られたテーブル出力あるいは乱数を，次の乗算器と加算器で処理します．テーブル出力を用いる場合は直線補間を，さらに波や波紋，砂嵐の場合には*m_gain*を乗算することになります．効果全般で，テーブルを参照するときには波長(半径)＝1に正規化されているので，元に戻した際に「波長＝R」となって表現されます．しかし，波や波紋においては，波高もテーブル上の値域にRをかけたものになってしまい，波高を独立して制御できません．そのため，*m_gain*の乗算によって波長と波高を独立して制御できるようにしてあります．乱数を使用する場合，*m_gain*の乗算のみで直線補間は行われません．このようにして，変形成分が生成されます．

● 下段で非変形成分を生成する

図15のブロック図の下段は，変形を伴わず直線性を保ったまま最終段に座標を送るパスです．

ただし，入力段には丸め回路が入っています．これは，小数以下を丸めてモザイクを生成する場合に使用されます．機能的には変形成分を生成する中段の機能となりそうなのですが，y座標が通過するのはここだけなので，ここでの処理としています．

中段と下段によって変形成分と非変形成分が生成されたので，**表3**の式に従って加減算を行い，非線形演算の処理を終了します．

各種定数の足し込みもここで行っています．半径が1に正規化されているので，定数も半径の大きさに合わせることなく，それぞれ1種類で済むことになります．

● 20万システム・ゲート中60％程度の使用率に抑える

ここまで述べてきた回路を実現するため，実際にVerilog HDLでコーディングしてみました．このRTLを米国Synplicity社のSynplify Pro Ver.7.2で論理合成した結果の

図16　アドレス生成部の論理合成結果

```
                    Requested       Estimated
Starting Clock      Frequency       Frequency
-----------------------------------------------
ss_adgena|clk       110.0 MHz       117.0 MHz
===============================================
Resource Usage Report for ss_adgena
Mapping to part: xc3s400pq208-4

Register bits not including I/Os: 1915 (26%)

Mapping Summary: Total  LUTs: 2394 (33%)

RAM/ROM usage summary
64x1 ROMs (ROM64X1): 36
256x1 ROMs (ROM256X1): 15
Block Rams : 2 of 16 (12%)

Block Multipliers: 8 of 16 (50%)
```

図17　ビデオ信号処理などを含めた回路の論理合成結果

```
                    Requested       Estimated
Starting Clock      Frequency       Frequency
-----------------------------------------------
dve2d|clk027i        28.0 MHz        35.3 MHz
dve2d|dcm0_clkfx    112.0 MHz       114.6 MHz
===============================================
Resource Usage Report for dve2d
Mapping to part: xc3s400pq208-4

Register bits not including I/Os: 3742 (52%)

Mapping Summary: Total  LUTs: 4118 (57%)

RAM/ROM usage summary
Dual Port Rams (RAM16X1D): 21
64x1 ROMs (ROM64X1): 36
Block Rams : 3 of 16 (18%)

Block Multipliers: 13 of 16 (81%)
```

レポートを**図16**に示します．かなり小さくまとめられたと思いますが，いかがでしょうか．もちろん，アドレス生成部は単独ではなんの意味も持たないので，ビデオ信号処理部（前処理フィルタは省略）とSDRAMを4個使用したメモリ・コントローラを含めた結果を**図17**に示します．

実は，**図1**などで示した効果のサンプル画像には，ここまで説明した画像の変形効果のほかに，光の反射と影も付けてあります．これによって2Dでありながら立体的に見せているのですが，今回はここまで紹介できませんでした．合成結果のレポートには，この機能回路も含まれていることを付け加えておきます．

また，Xilinx社の「Spartan-3 Starter Kit」に付属しているボードを用いて，実機動作も行いました．残念ながら，画像の入出力ボードを作る時間的な余裕がなかったので，ソース画面はFPGA内部で生成した固定パターンです（変形した画像が動画であるからこそのハードウェア実現なのだが…）．おかげでというわけではないのですが，ビデオ系の処理回路がほとんどないので，ボードに実装されている20万システム・ゲートのFPGA（Spartan-3 XC3S200）の60％程度の使用率で収めることができました．また，画像出力機能がないので，ボードに付いているVGA出力ポートを使用して画像を出力しています（8色しか表示できないので，**写真1**のような単調な模様しか出ない）．

システムをスマートにまとめようとした場合，とくに演算回路においては，どの演算をどのようにハードウェアに割り当てるか，といった演算回路そのものではない部分でのくふうがかなり重要です．ハードウェアに割り当てられてからの回路上のくふうで行える範囲というのは意外に限られており，システム・レベルから見れば重箱の隅をつつく程度です．

ソフトウェアとの協調をうまくやる，アプリケーションの特性を生かす，数学的な最適化を行うなど，回路設計前にやっておくことはきわめて重要です．そして，このような段取りを成功させるために必要なことは，仕様をきちんと構築してからシステム・レベルの最適化を始めることです．最適化の工程が進んでからの仕様変更は，プロジェクトに多大なダメージを与えます．まして，なんとなく設計を始めて，そのつど仕様を足していくというのは論外です．

もっとも，理想どおりにことが運ばないというのは，みなさんもよくご存じのことでしょう．そうした現状の渦に巻き込まれないようにすることは，人の努力以外にはありえません（こういうことを解決できるツールがあったらいいのになぁ…）．日々，心がけたいものです．

参考・引用*文献

(1) David F. Rogers，J. Alan Adams；コンピュータグラフィックス，日刊工業新聞社，1979年1月．
(2) 小林芳直；浮動小数点演算回路を極める，Design Wave Magazine，pp.44-56，2003年7月号．
(3) 鈴木昌治；ASIC開発における数値演算回路設計の定石，Design Wave Magazine，pp.24-38，2003年7月号．
(4) 近藤 渉；Spartan-3ファミリの機能，Design Wave Magazine，pp.35-41，2005年1月号．

はすとり・あすか

<筆者プロフィール>
蓮鳥明日香．酒を飲み，仲間と語らうのが，何よりも楽しみなのですが，つい酒がすぎることも．人生を飲まれないように気をつける今日このごろです．

3